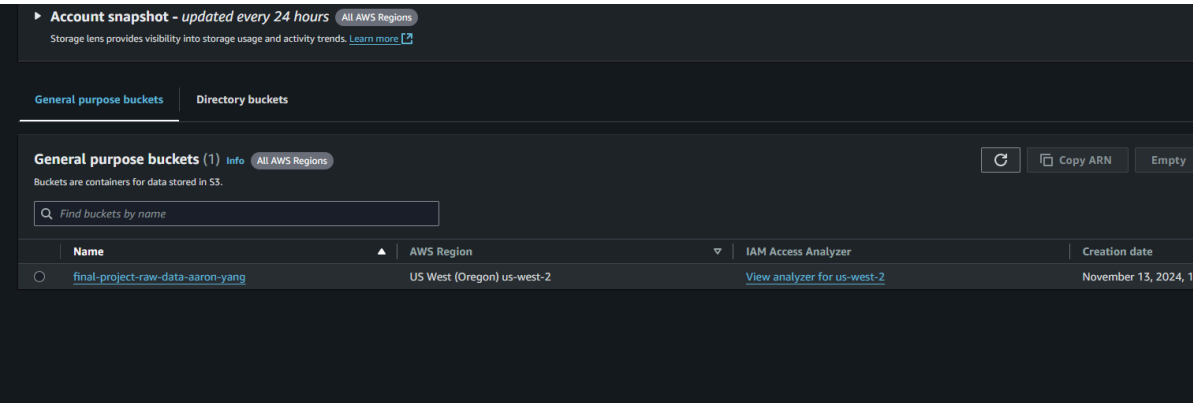


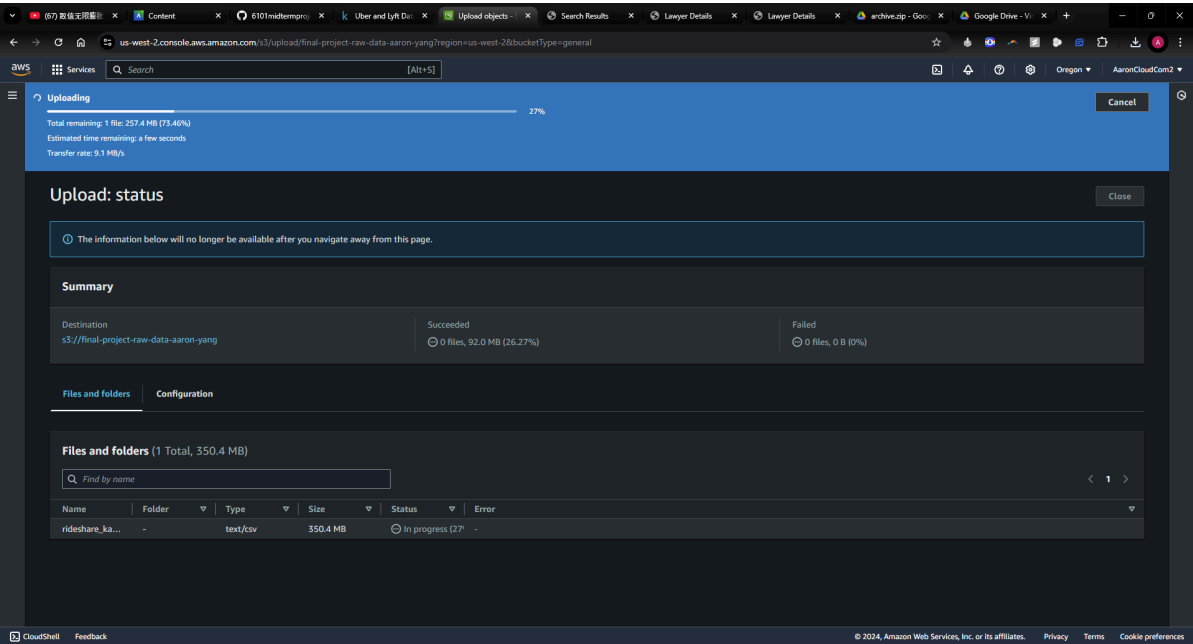
Final Project for Cloud Computing

Step 1 - Data Storage with Amazon S3

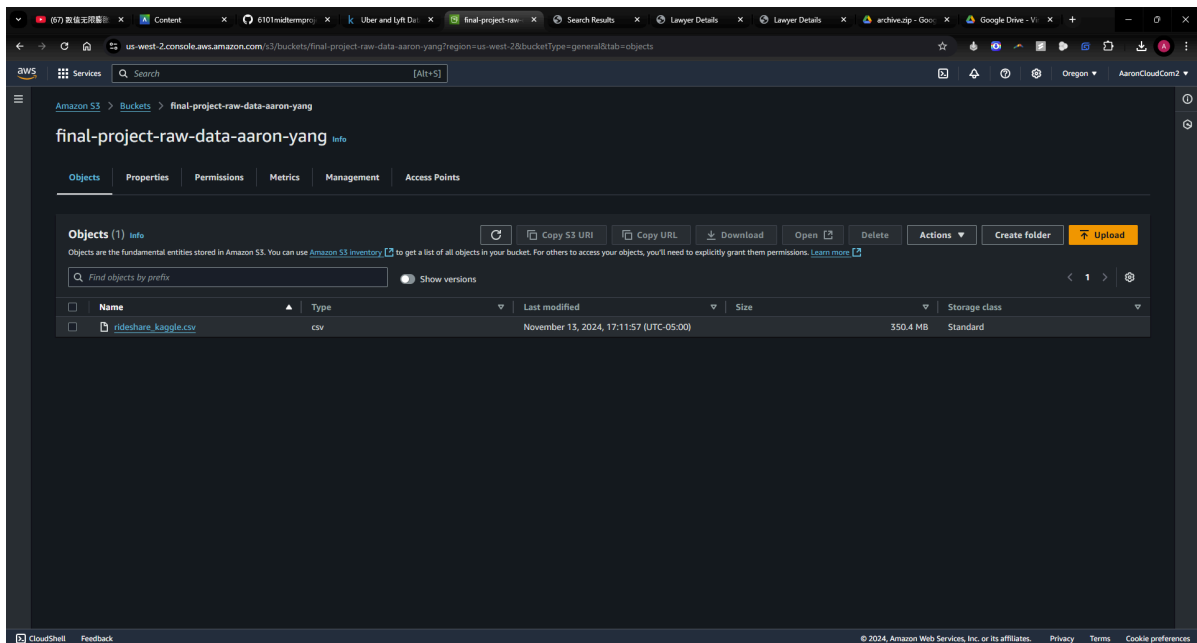
1. Create Amazon S3



2. Upload dataset into Amazon S3

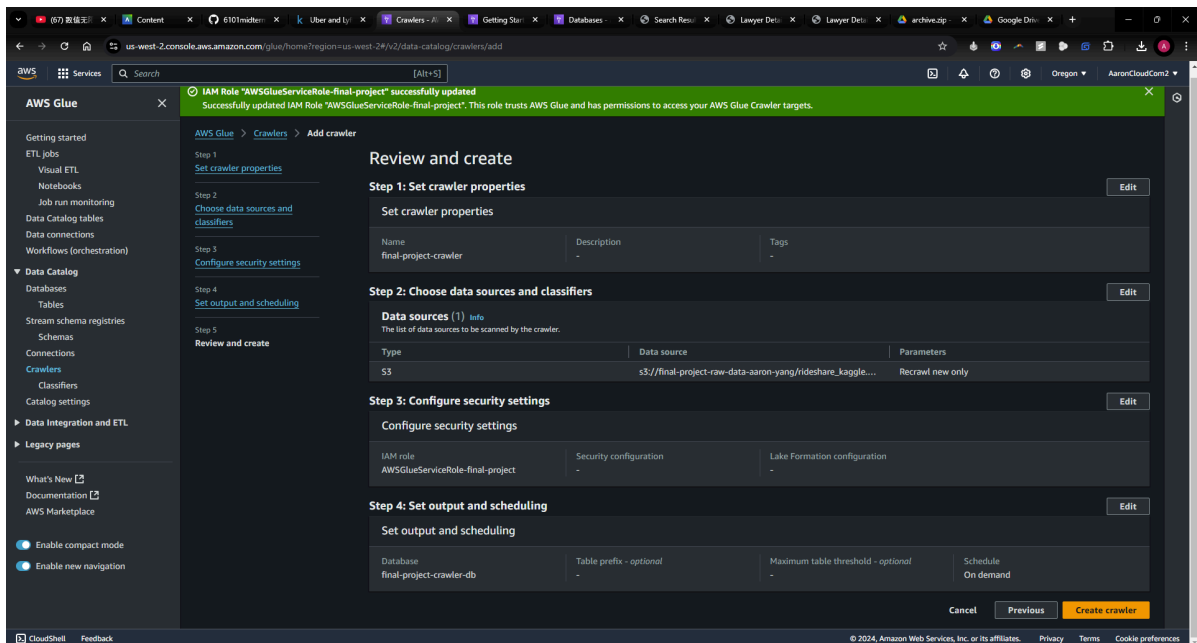


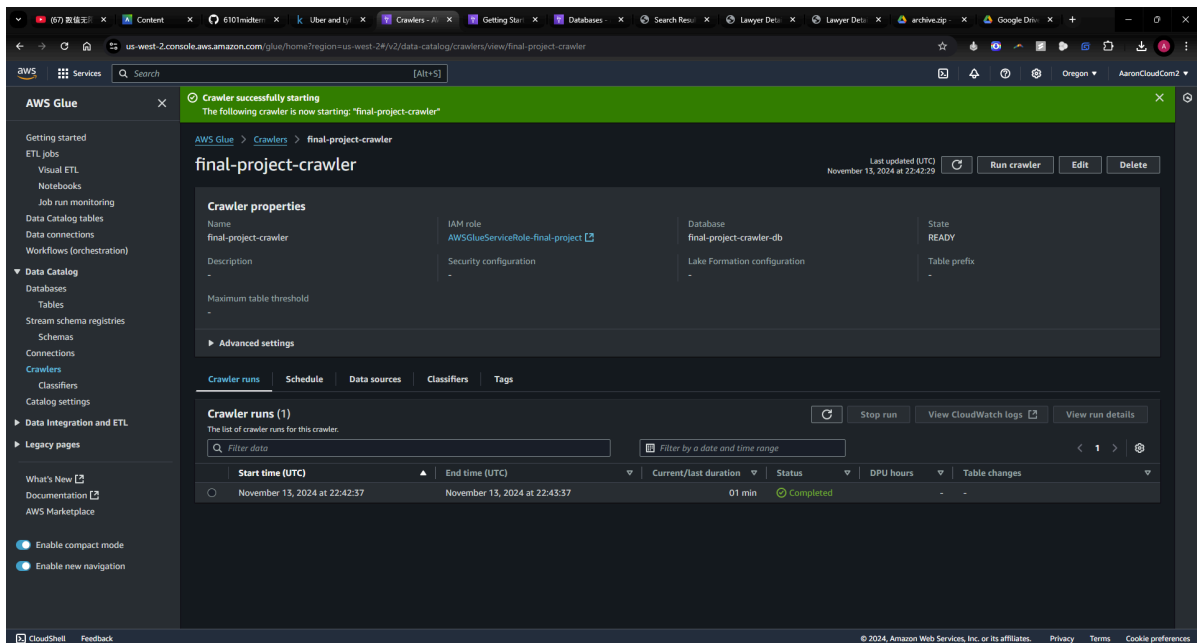
3. Result



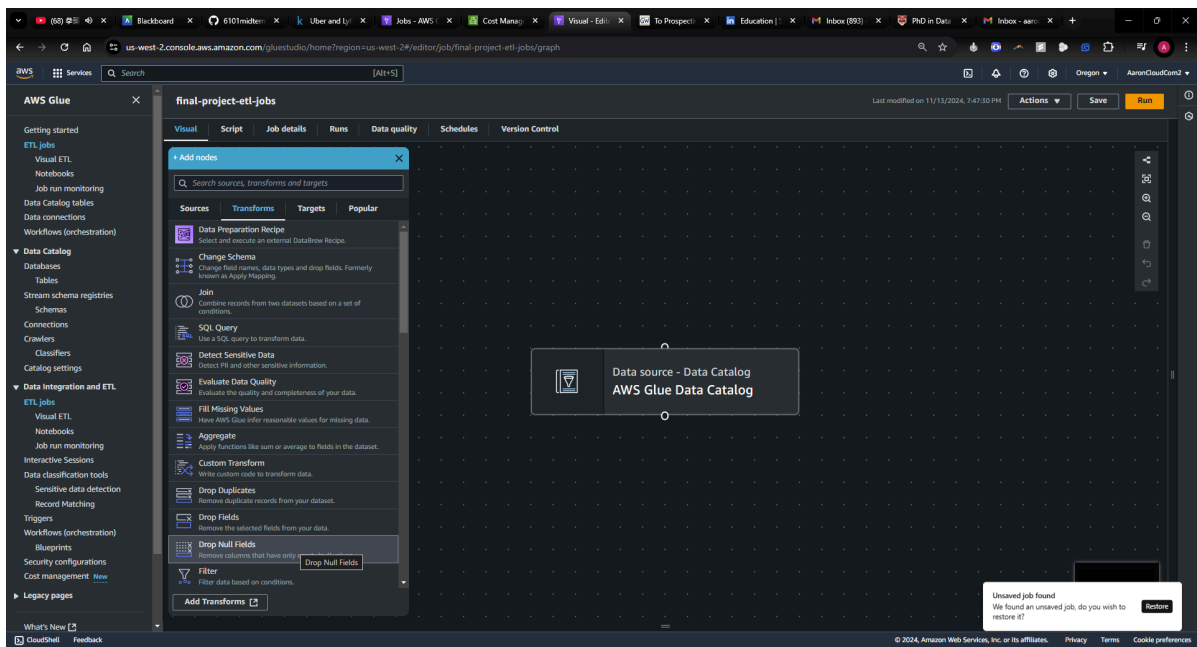
Step 2 - Data Cleaning with AWS Glue

1. Crawler Setup



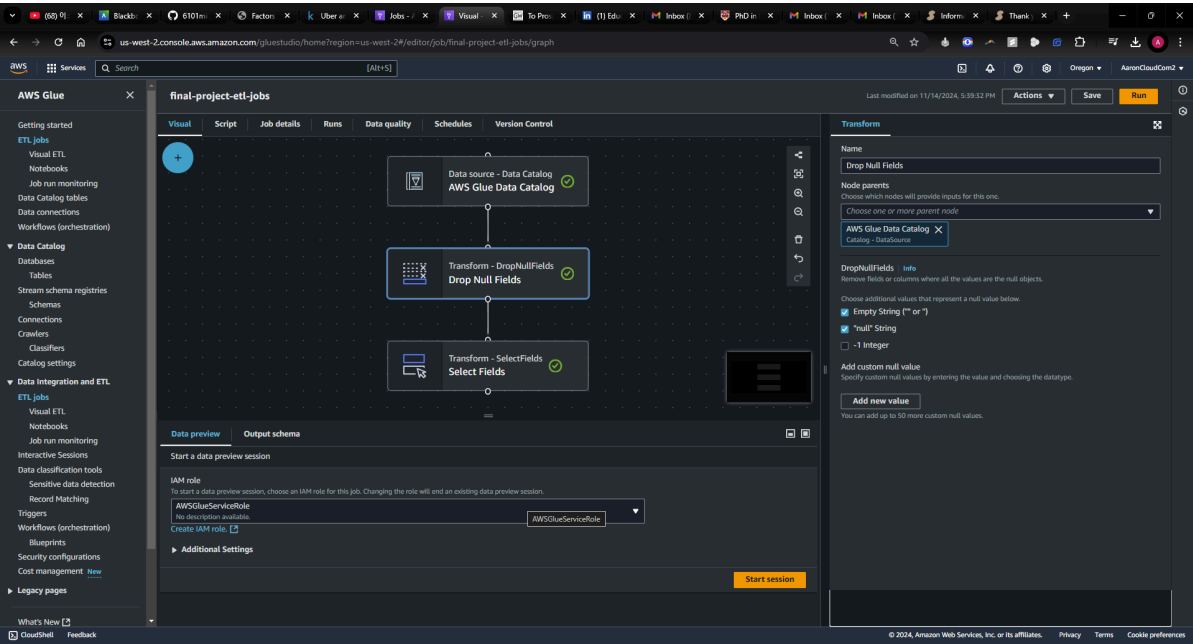


2. Create Jobs

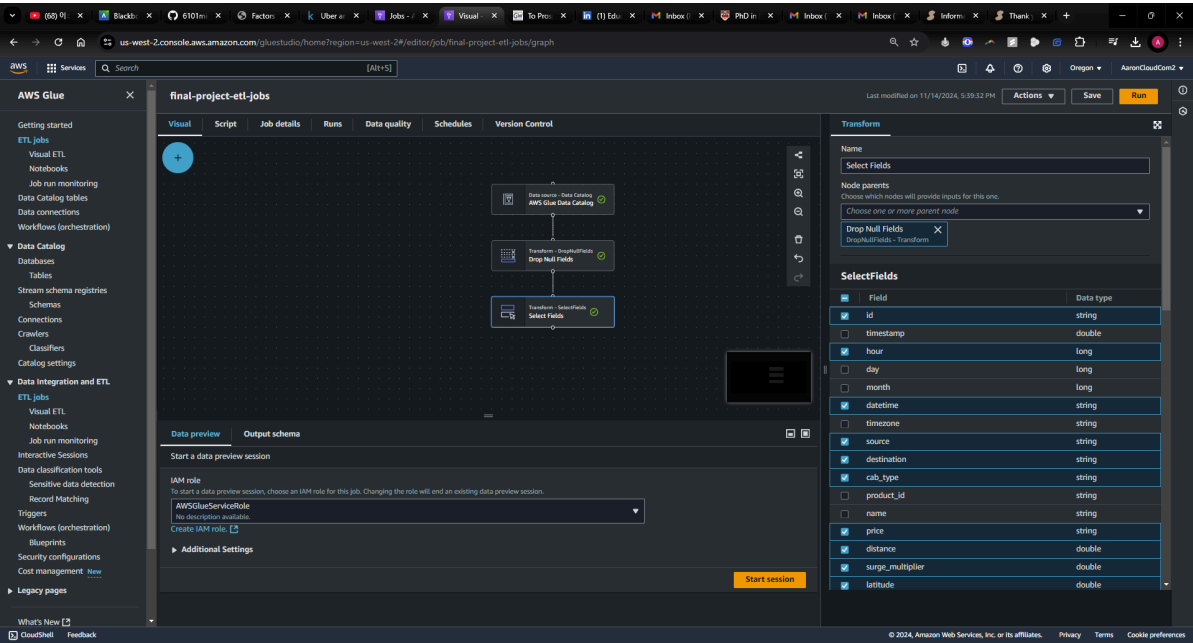


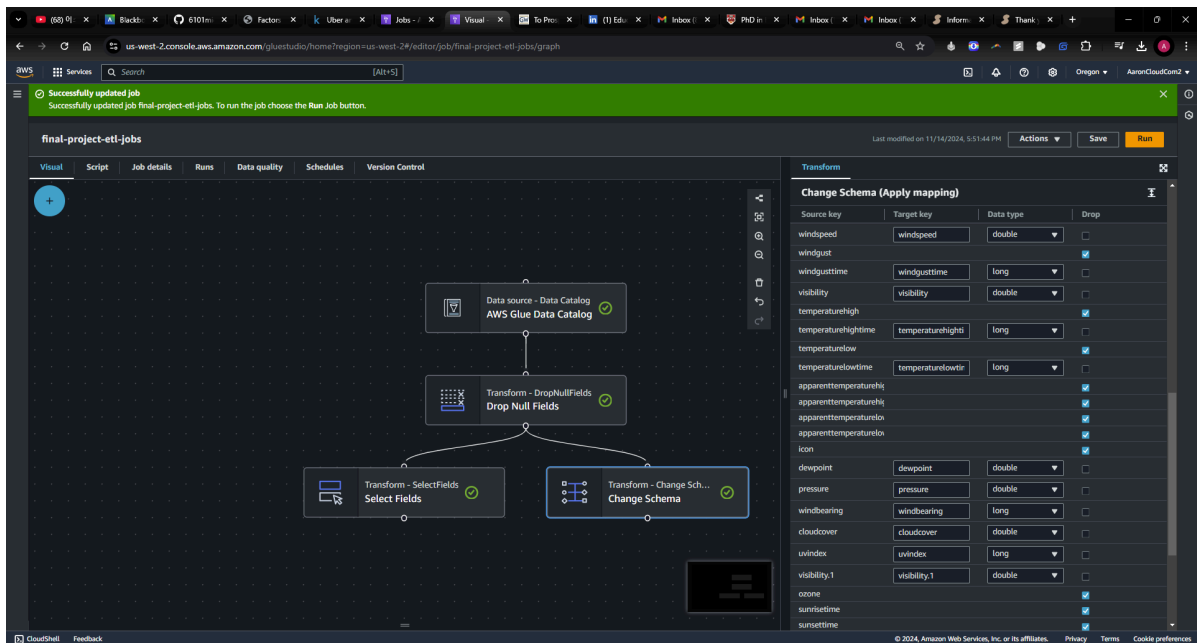
3. Cleaning data with AWS Glue

3.1 Drop null values and empty string

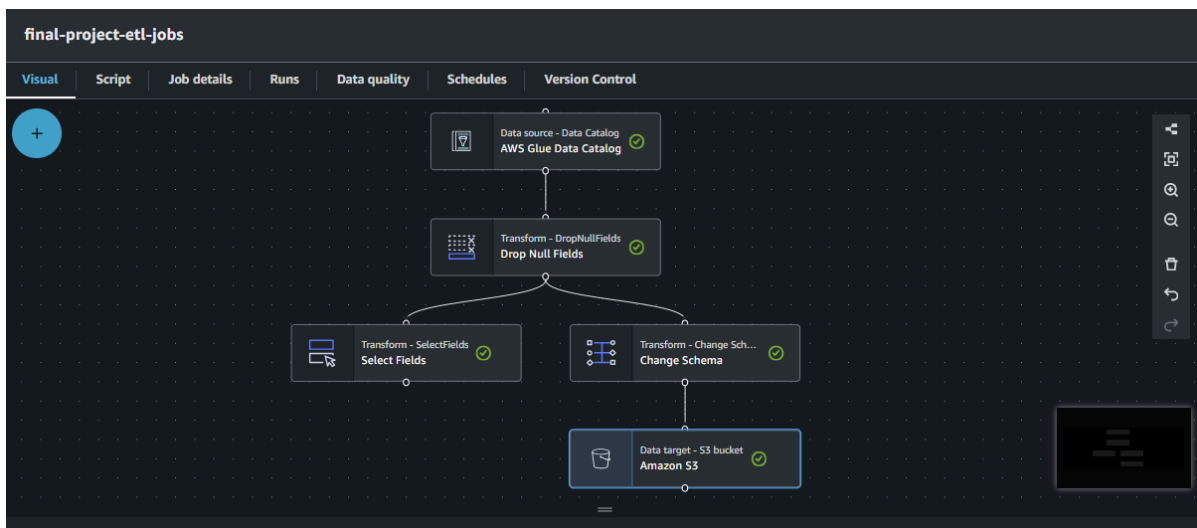


3.2 Select features through domain knowledge



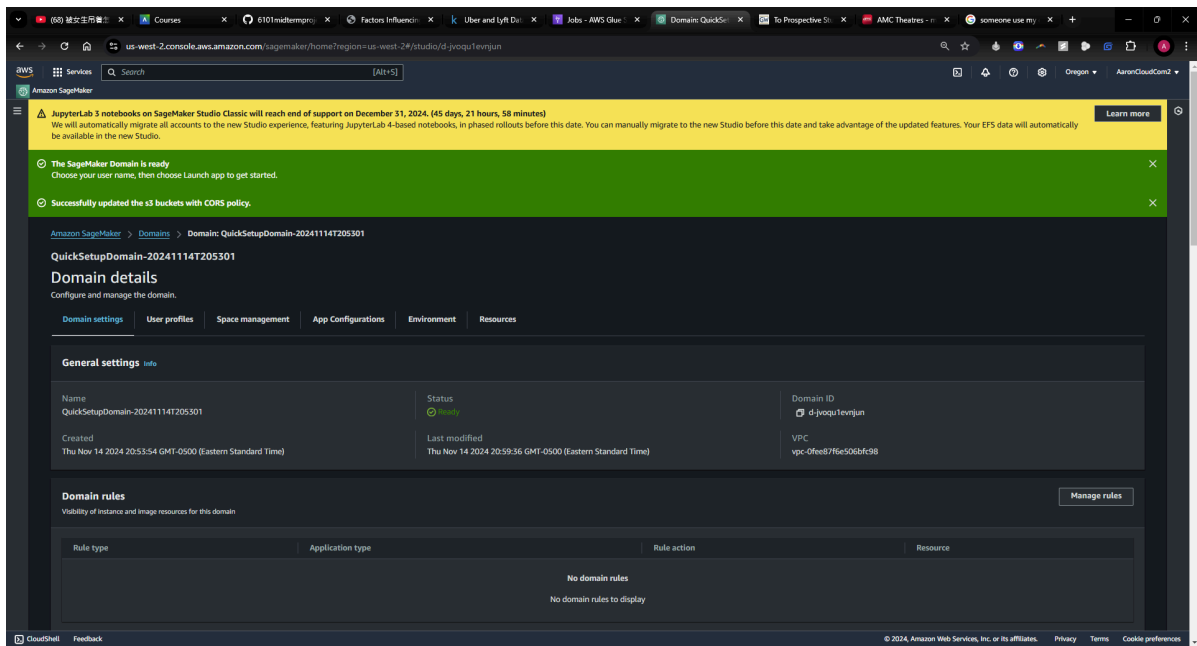


3.3 Set up the S3 Target

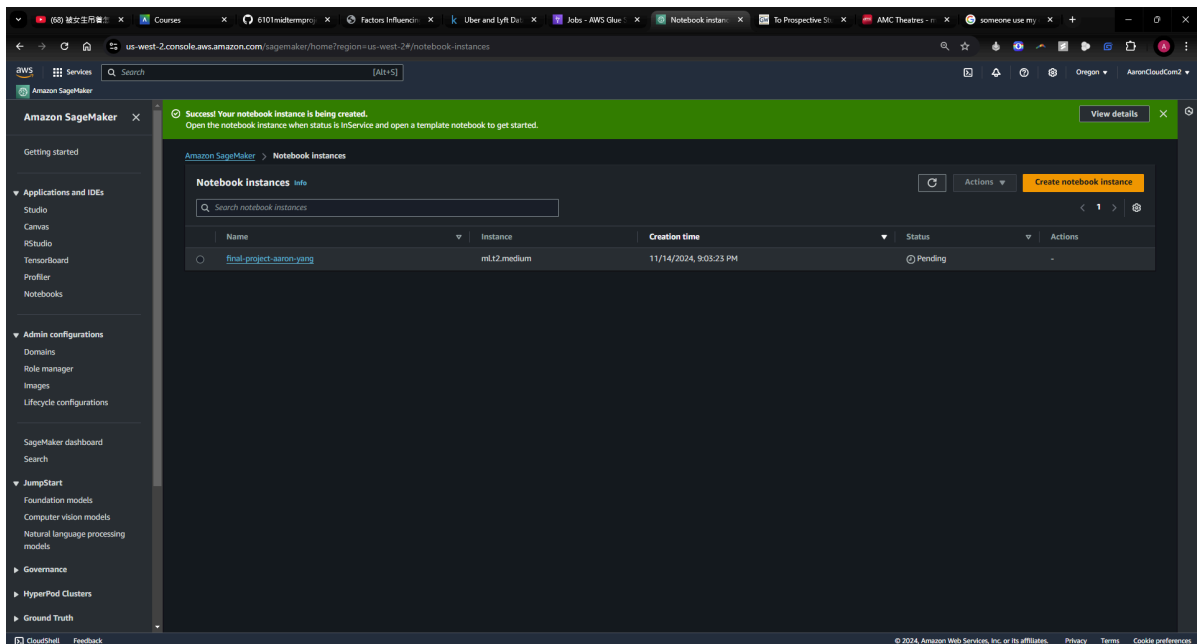


Step 3 - Model Training with Amazon SageMaker

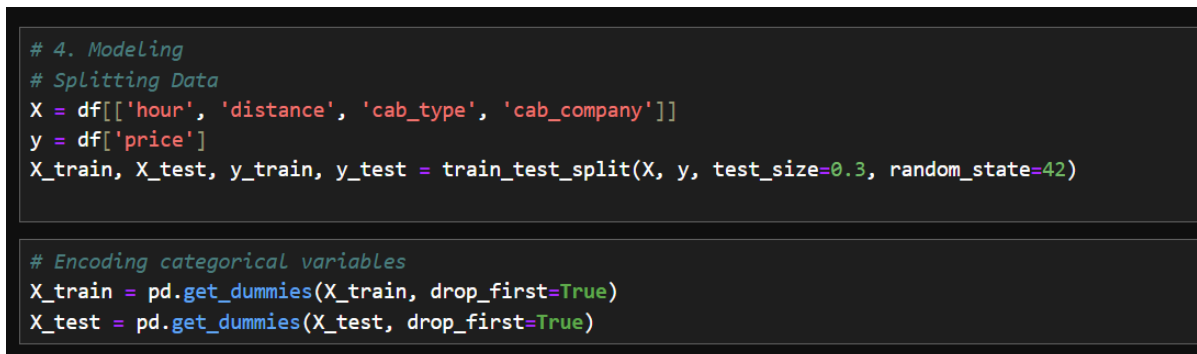
3.1 Create a sagemaker



3.2 Create a notebook



3.3 Model training



Linear Model Evaluation

R2 Score: 0.8928812065972889, MSE: 9.34479501014709

PCR Model Evaluation

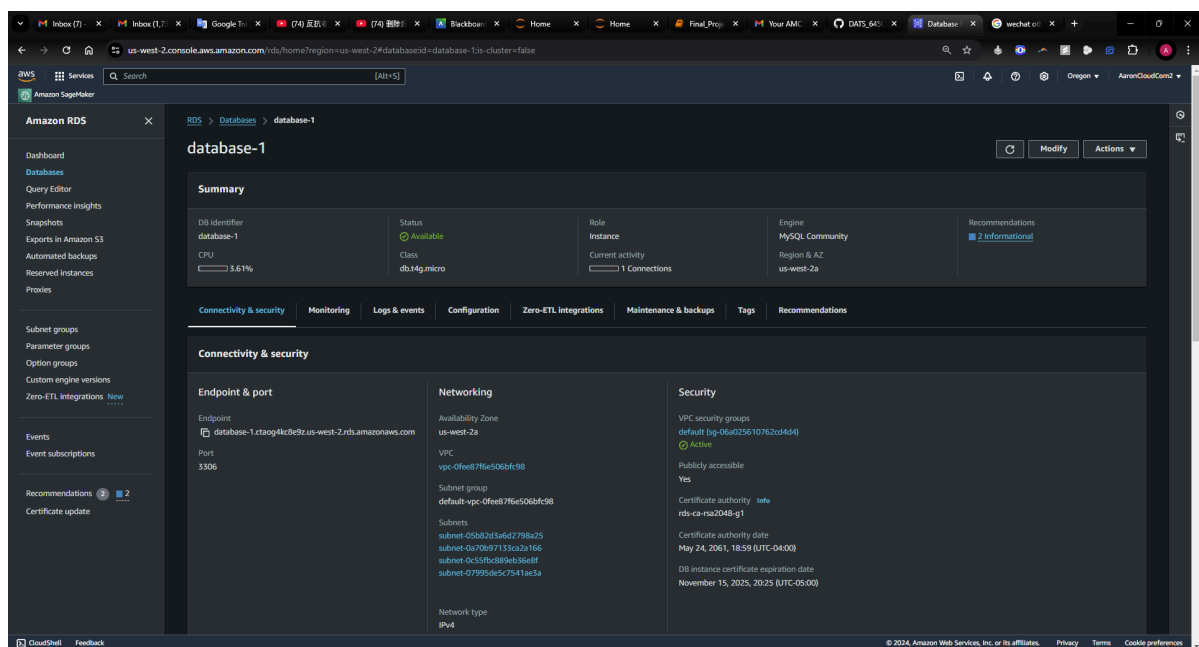
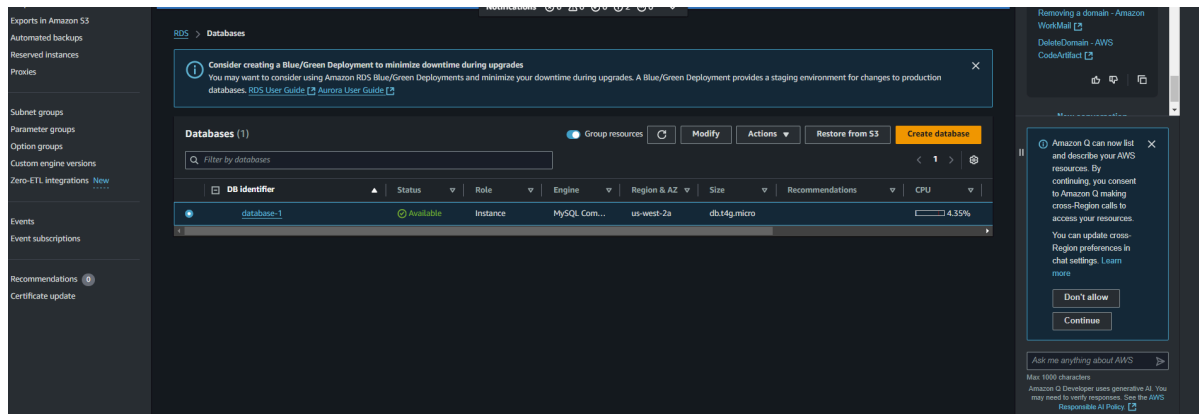
R2 Score: 0.4197360145084884, MSE: 50.62088382384545

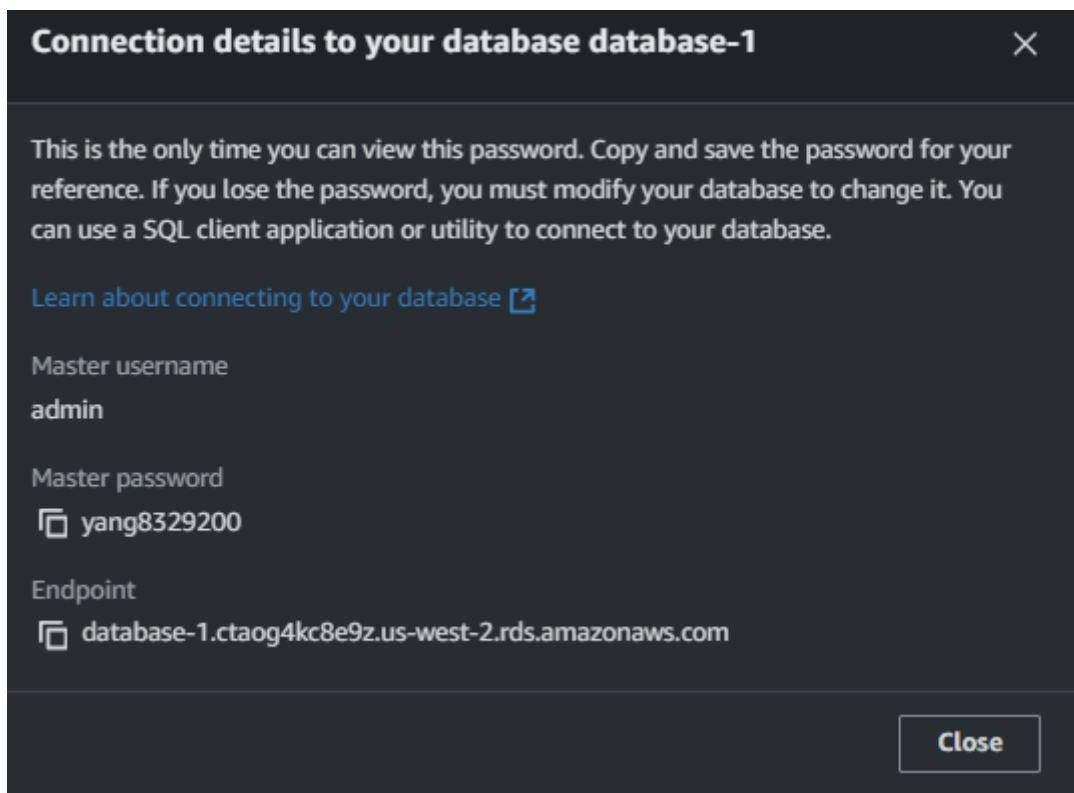
Decision Tree Evaluation

R2 Score: 0.9000511708682163, MSE: 8.71930396218501

Step 4 - Storing Results with Amazon RDS (Relational Database Service)

4.1 Create database





```
import pymysql

# Connect to the RDS database
connection = pymysql.connect(
    host='database-1.ctaog4kc8e9z.us-west-2.rds.amazonaws.com',
    port=3306,
    user='admin',
    password='yang8329200',
)

# Create a database and use it
with connection.cursor() as cursor:
    cursor.execute("CREATE DATABASE IF NOT EXISTS database_1;")
    cursor.execute("USE database_1;")

# Create a table for predictions
with connection.cursor() as cursor:
    create_table_query = """
    CREATE TABLE IF NOT EXISTS predictions (
        id INT AUTO_INCREMENT PRIMARY KEY,
        hour INT,
        distance FLOAT,
        cab_type VARCHAR(255),
        cab_company VARCHAR(255),
        predicted_price FLOAT,
        actual_price FLOAT
    );
    """
    cursor.execute(create_table_query)
    connection.commit()

# Insert predictions into the table
with connection.cursor() as cursor:
```



```

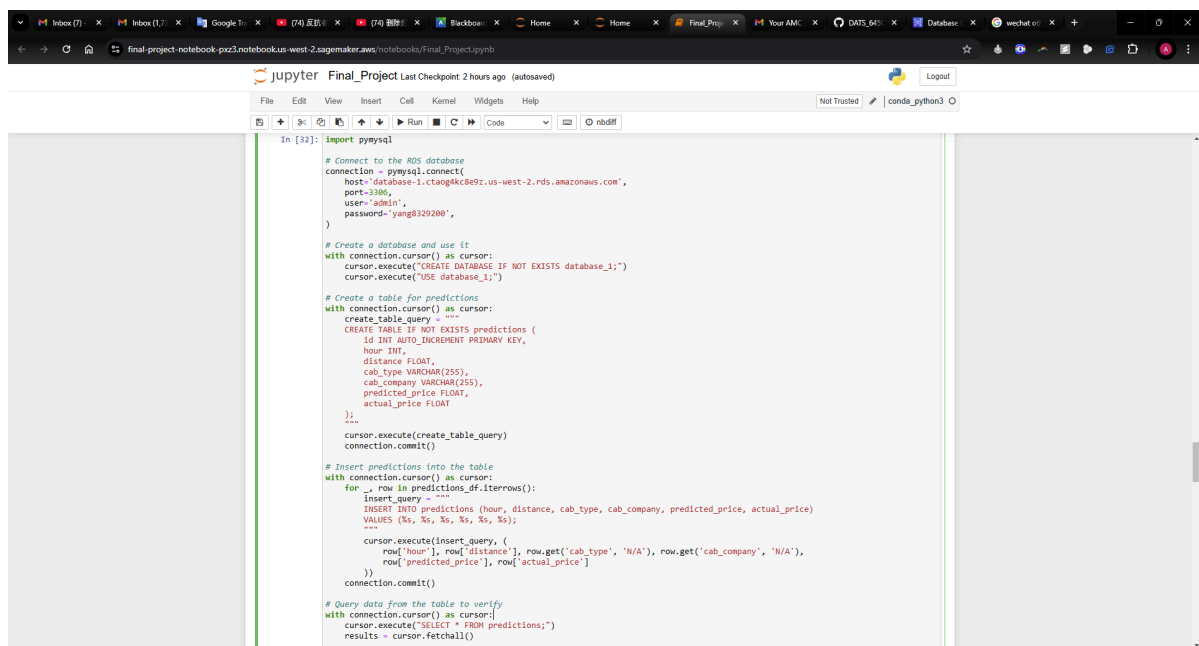
for _, row in predictions_df.iterrows():
    insert_query = """
        INSERT INTO predictions (hour, distance, cab_type, cab_company,
predicted_price, actual_price)
        VALUES (%s, %s, %s, %s, %s, %s);
    """

    cursor.execute(insert_query, (
        row['hour'], row['distance'], row.get('cab_type', 'N/A'),
row.get('cab_company', 'N/A'),
        row['predicted_price'], row['actual_price']
    ))
    connection.commit()

# Query data from the table to verify
with connection.cursor() as cursor:
    cursor.execute("SELECT * FROM predictions;")
    results = cursor.fetchall()

# Close the connection
connection.close()

```



The screenshot shows a Jupyter Notebook titled 'Final_Project' with a 'conda_python3' kernel. The code in the notebook is as follows:

```

In [32]: import pymysql

# Connect to the RDS database
connection = pymysql.connect(
    host='database-1.ctagk8cc8e9z.us-west-2.rds.amazonaws.com',
    port=3306,
    user='admin',
    password='yang8329200',
)

# Create a database and use it
with connection.cursor() as cursor:
    cursor.execute("CREATE DATABASE IF NOT EXISTS database_1;")
    cursor.execute("USE database_1;")

# Create a table for predictions
with connection.cursor() as cursor:
    create_table_query = """
        CREATE TABLE IF NOT EXISTS predictions (
            id INT AUTO INCREMENT PRIMARY KEY,
            hour INT,
            distance FLOAT,
            cab_type VARCHAR(255),
            cab_company VARCHAR(255),
            predicted_price FLOAT,
            actual_price FLOAT
        );
    """
    cursor.execute(create_table_query)
    connection.commit()

# Insert predictions into the table
with connection.cursor() as cursor:
    for _, row in predictions_df.iterrows():
        insert_query = """
            INSERT INTO predictions (hour, distance, cab_type, cab_company, predicted_price, actual_price)
            VALUES (%s, %s, %s, %s, %s, %s);
        """
        cursor.execute(insert_query, (
            row['hour'], row['distance'], row.get('cab_type', 'N/A'), row.get('cab_company', 'N/A'),
            row['predicted_price'], row['actual_price']
        ))
    connection.commit()

# Query data from the table to verify
with connection.cursor() as cursor:
    cursor.execute("SELECT * FROM predictions;")
    results = cursor.fetchall()

```

```
);
...
cursor.execute(create_table_query)
connection.commit()

# Insert predictions into the table
with connection.cursor() as cursor:
    for _, row in predictions_df.iterrows():
        insert_query = """
        INSERT INTO predictions (hour, distance, cab_type, cab_company, predicted_price, actual_price)
        VALUES (%s, %s, %s, %s, %s, %s);
        """
        cursor.execute(insert_query, (
            row['hour'], row['distance'], row.get('cab_type', 'N/A'), row.get('cab_company', 'N/A'),
            row['predicted_price'], row['actual_price']
        ))
        connection.commit()

# Query data from the table to verify
with connection.cursor() as cursor:
    cursor.execute("SELECT * FROM predictions;")
    results = cursor.fetchall()

# Close the connection
connection.close()

(7, 17, 3.4, 'N/A', 'N/A', 18.7374, 19.5)
(8, 6, 0.73, 'N/A', 'N/A', 13.6376, 11.0)
(9, 12, 2.45, 'N/A', 'N/A', 18.4959, 11.0)
(10, 23, 2.8, 'N/A', 'N/A', 32.007, 28.5)
(11, 9, 2.02, 'N/A', 'N/A', 11.5988, 9.5)
(12, 9, 5.7, 'N/A', 'N/A', 19.7166, 16.0)
(13, 0, 1.16, 'N/A', 'N/A', 5.83889, 6.5)
(14, 19, 1.57, 'N/A', 'N/A', 7.59986, 7.5)
(15, 10, 2.69, 'N/A', 'N/A', 7.45875, 7.0)
(16, 4, 2.27, 'N/A', 'N/A', 9.9876, 9.5)
(17, 4, 3.44, 'N/A', 'N/A', 24.6517, 26.0)
(18, 3, 4.72, 'N/A', 'N/A', 13.2883, 16.5)
(19, 11, 2.37, 'N/A', 'N/A', 32.8254, 27.5)
(20, 2, 3.22, 'N/A', 'N/A', 33.2831, 30.5)
(21, 10, 2.4, 'N/A', 'N/A', 18.3716, 16.5)
(22, 23, 1.79, 'N/A', 'N/A', 8.62885, 8.5)
(23, 13, 3.8, 'N/A', 'N/A', 14.3234, 11.5)
(24, 3, 2.39, 'N/A', 'N/A', 32.8841, 30.0)
(25, 2, 2.33, 'N/A', 'N/A', 10.1582, 9.5)
Row (25, 2, 2.33, 'N/A', 'N/A', 10.1582, 9.5)

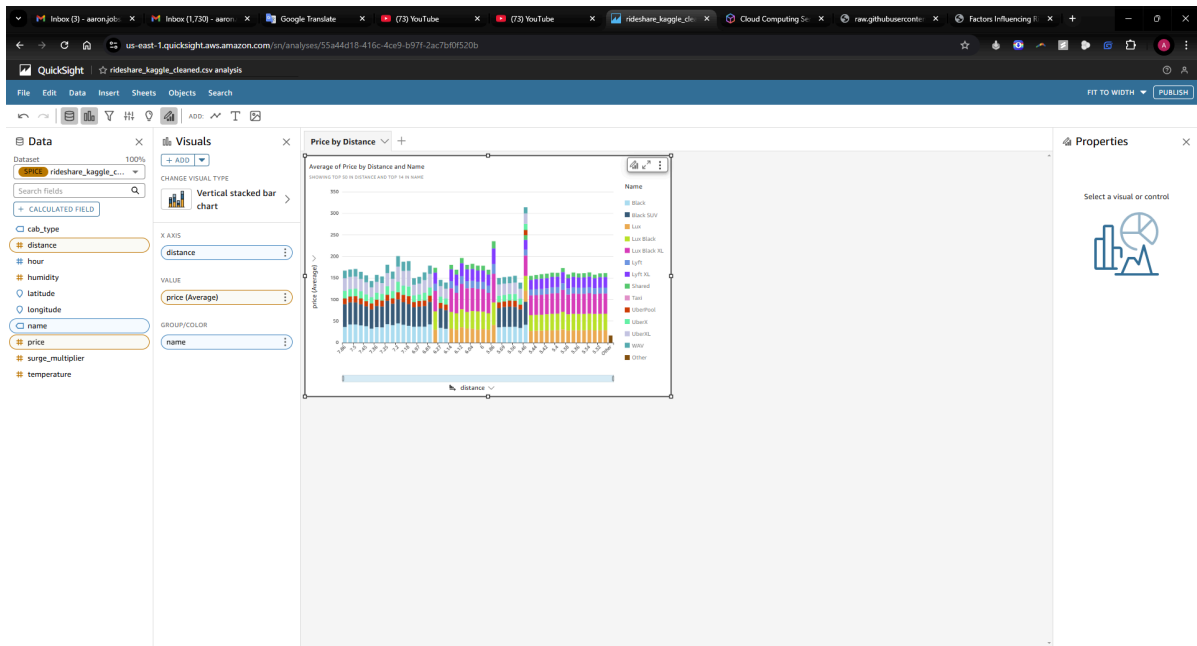
In [20]: PCA for Principal Component Regression
```

Step 5 - Data Visualization with Amazon QuickSight

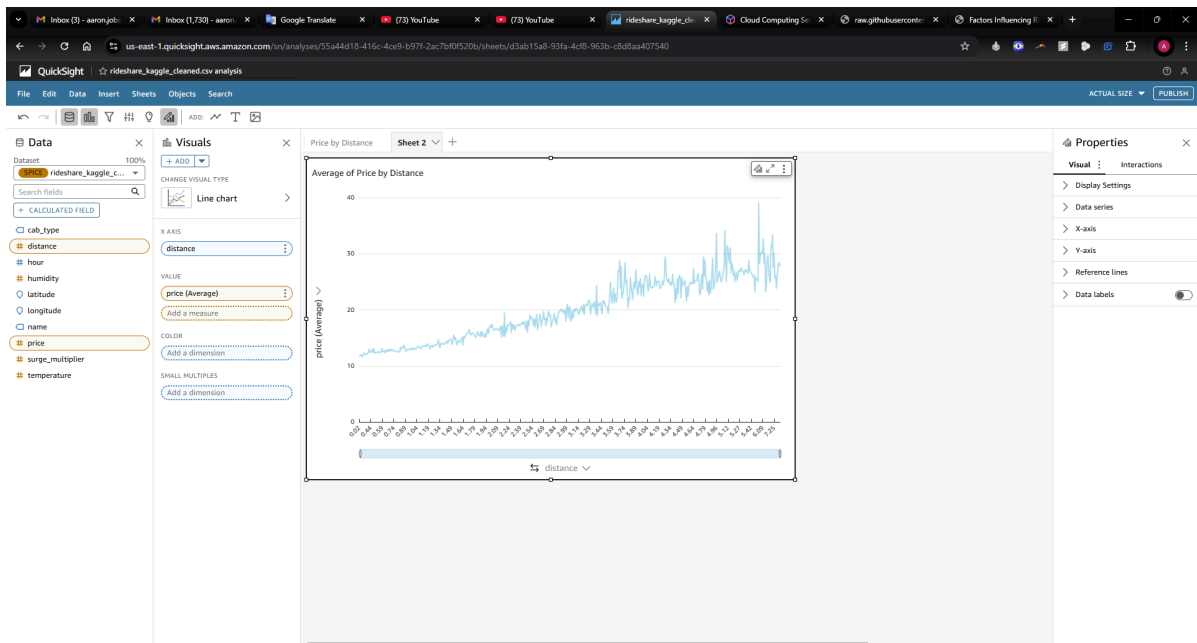
5.1

5.2 Graphs

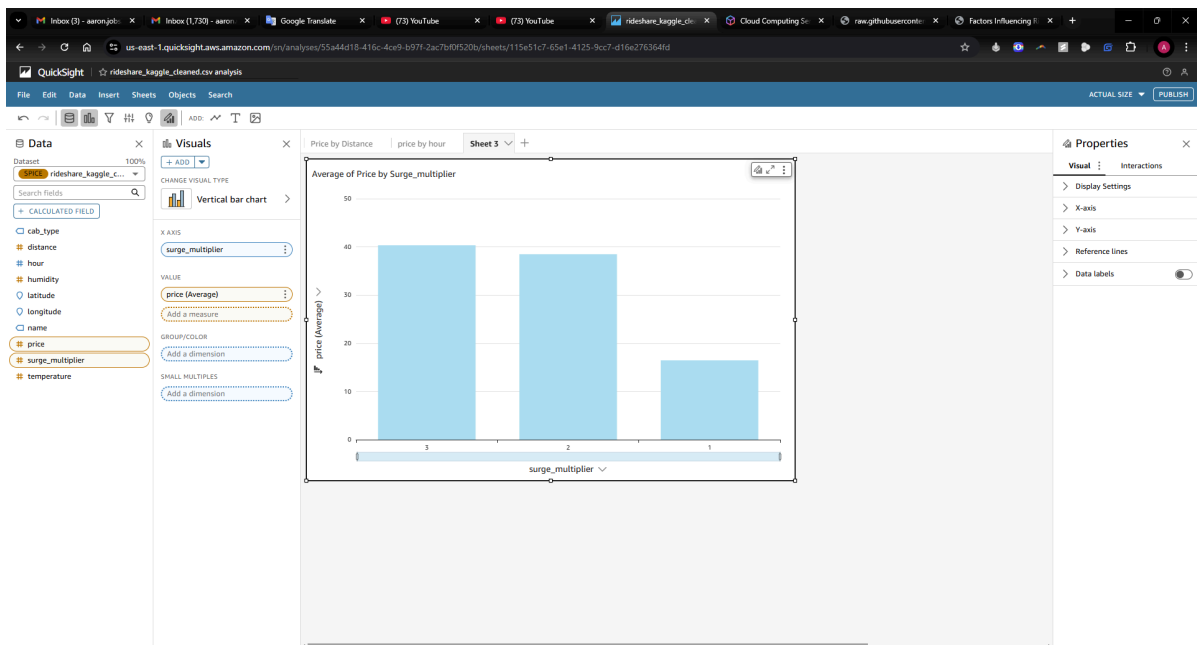
Price by Distance by cab_type



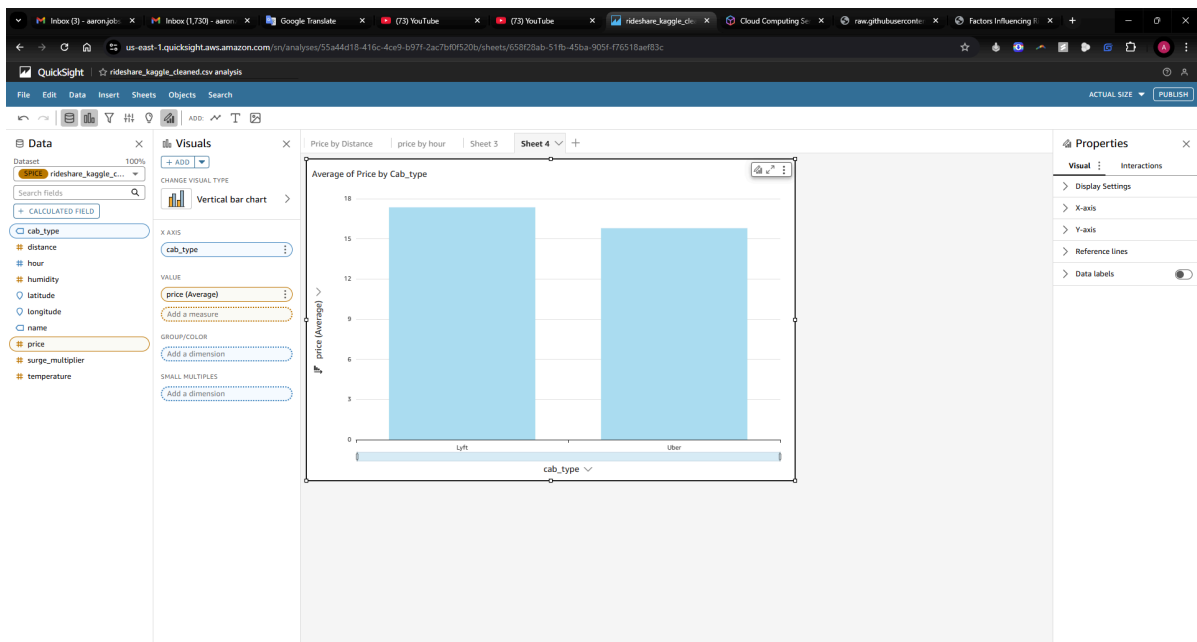
price by distance



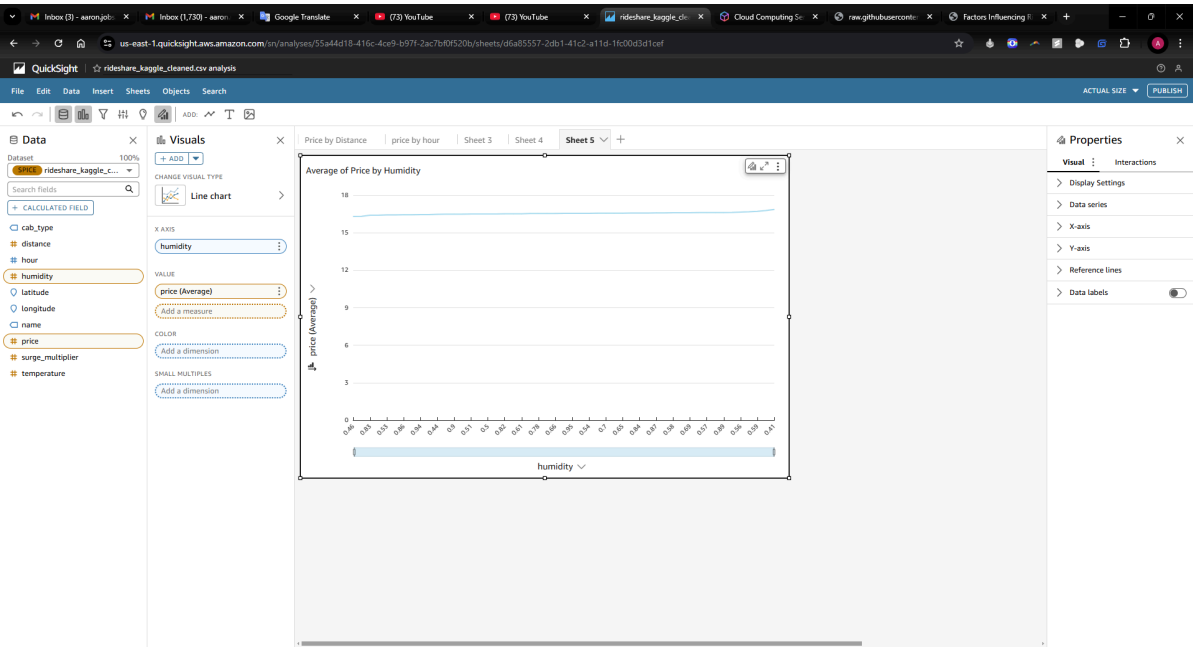
price by surge_multiplier



price by company



price by humidity



price by temperature

